Image Recognition: Learning to Distinguish Ships from Icebergs

Sai Surya Teja Maddikonda
Northeastern University, Boston, MA
maddikonda.s@husky.neu.edu

Abstract

The aim of this project was to achieve successful recognition of images by experimenting with some of the widely used machine learning / deep learning algorithms in the field of image classification. The idea was to apply image recognition in the context of distinguishing the images of ships from the images of icebergs. The image data does not consist of the conventional non-negative integral RGB values, but are float numbers with units as decibels (dB), and bear physical meaning, as characterized by the radar backscatter produced from different polarizations at a particular angle of incidence. Hence, meticulous data preparation was necessary. Three algorithms were considered. k-Nearest Neighbors was chosen as a baseline method and was experimented with varying the values of number of neighbors (k). Random Forests was tried with the intention of experimenting on an ensemble learner by varying the number of estimators. Finally, two models of Convolutional Neural Networks (CNNs) were explored which differ in the number of convolution layers, dropout layers and batch sizes. Upon training these models on various inputs that differed in the data preprocessing steps, CNN models offered the best accuracy among all.

Introduction

Drifting icebergs present threats to navigation and activities in areas such as offshore of the East Coast of Canada. Currently, many institutions and companies use aerial reconnaissance and shore-based support to monitor environmental conditions and assess risks from icebergs. However, in remote areas with particularly harsh weather, these methods are not feasible, and the only viable monitoring option is via satellite.

There have been ongoing endeavors by companies such as StatOil, an international energy company operating worldwide, in collaboration with satellite data research agencies like C-CORE, to build a computer vision based surveillance system using satellite imagery. To keep operations safe and efficient and cost effective, the intentions of these institutions is to leverage machine learning to more accurately detect and discriminate against threatening icebergs as early as possible.

Identifying and discriminating against icebergs with the aid of machine learning necessitates building a machine learning algorithm that is capable of efficiently classifying an unseen image of a ship or an iceberg. This requirement falls in the image recognition / image classification area of computer vision, where the learning model is tasked with determining whether or not the image data contains some specific object, feature, or activity. determining whether or not the image data contains some specific object, feature, or activity.

This task of recognizing a visual concept is relatively trivial for a human to perform, but it is worth considering the challenges involved from the perspective of a computer vision algorithm: Viewpoint Variation (orientation of image with respect to the camera/sensor), Scale Variation (variation in size of the visual classes), Deformation (rigid bodies vs. deformed bodies), Occlusion (little visibility of the object of interest), Illumination Conditions, Background Clutter, Intra-Class Variation (broad classes of interest), etc. A good image classification model must be invariant to the cross product of all these variations, while simultaneously retaining sensitivity to the inter-class variations.

Having established that the task in image classification is to take an array of pixels that represents a single image and assign a label to it, the complete pipeline can be formalized as follows.

Input: The input (training set) consists of a set of N images, each labelled with one of the K classes (like ship, iceberg).

Training: The learning model (classifier algorithm) uses the training set to learn what every one of the classes looks like.

Evaluation: In the end, the quality of the model is evaluated by making it predict the class labels for unseen images, and then comparing these predictions with the ground-truth.

In this project, the first, simple, and baseline method considered for implementing image classification is k-Nearest Neighbor classifier, which operates by comparing the unseen test image with the each of the labelled training images, and predicting the class label based on the k closest training images. As a second model, the Random Forest classifier model was considered, which is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. Finally, as the main method of this project, the Convolutional Neural Network (CNN) model was explored. Two variants of CNNs with different hyperparameters were built. CNNs have proven to be one of the best algorithms for the image classification task, with their capabilities close to humans (Russakovsky et al. 2015), as illustrated by their performance on datasets such as ImageNet Large Scale Visual Recognition. These algorithms are explained in greater detail in the next section.

Background

Background information about k-Nearest Neighbors, Random forests and Convolutional Neural Networks classifier models is presented here.

k-Nearest Neighbors

The k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression (Altman 1992). In both cases, the input consists of the k closest training examples in the feature space. When k-NN is used for classification task, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

The training examples are vectors in a multidimensional feature space, each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and an unlabeled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point. A commonly used distance metric for continuous variables is Euclidean distance (or L_2 distance):

$$L_2(I_1, I_2) = \sqrt{\sum_p \left(I_1^p - I_2^p\right)^2}$$

Choosing the rectilinear (or L_1) distance is not uncommon:

$$L_1(I_1, I_2) = \sum_p \left|I_1^p - I_2^p\right|$$

A drawback of the basic 'majority voting' classification occurs when the class distribution is skewed. That is, examples of a more frequent class tend to dominate the prediction of the new example, because they tend to be common among the k nearest neighbors due to their large number (Coomans, Massart 1982). One way to overcome this problem is to weight the classification, taking into account the distance from the test point to each of its k nearest neighbors. The class (or value, in regression problems) of each of the k nearest points is multiplied by a weight proportional to the inverse of the distance from that point to the test point.

The best choice of k depends upon the data; generally, larger values of k reduces effect of the noise on the classification (Everitt et al. 2011), but make boundaries between classes less distinct. A good k can be selected by various heuristic techniques. The accuracy of the k-NN algorithm can be severely degraded by the presence of noisy or irrelevant features, or if the feature scales are not consistent with their importance.

Random Forests

Ensemble Learning in statistics and machine learning uses multiple learning algorithms to obtain better predictive performance than could be obtained from any of the constituent learning algorithms alone. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees (Ho 1995). Random decision forests correct for decision trees' habit of overfitting to their training set.

Decision trees are a popular method for various machine learning tasks. But trees that are grown very deep tend to learn highly irregular patterns: they overfit their training sets, i.e. have low bias, but very high variance. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance (Hastie et al. 2008). This comes at the expense of a small increase in the bias and some loss of interpretability, but generally greatly boosts the performance in the final model.

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set $X = x_1, \ldots, x_n$ with responses $Y = y_1, \ldots, y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For $b = 1, \ldots, B$:

1. Sample, with replacement, n training examples from X, Y; call these X_b, Y_b.
2. Train a classification or regression tree f_b on X_b, Y_b.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on x':

$$\hat{f} = \frac{1}{B}\sum_{b=1}^{B} f_b(x')$$

or by taking the majority vote in the case of classification trees. This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated.

The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called 'feature bagging'. The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated. An analysis of how bagging and random subspace projection contribute to accuracy gains under different conditions is given by (Ho 2002).

Convolutional Neural Networks (CNN)

Convolutional neural network (CNN, or ConvNet) is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. CNNs use a variation of multilayer perceptrons designed to require minimal preprocessing (LeCun 2013). They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function on the last (fully-connected) layer. CNN architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the number of parameters in the network.

CNN layer types mainly include three types, namely convolutional layer, pooling layer and fully-connected layer.

Convolutional Layer

Convolutional Layer is the core part of the CNN, which has local connection and weights of shared characteristics. The aim of Convolutional layer is to learn feature representations of the inputs. As shown in above, Convolutional layer consists of several feature maps. Each neuron of the same feature map is used to extract local characteristics of different positions in the former layer, but for single neurons, its extraction is local characteristics of same positions in former different feature map. In order to obtain a new feature, the input feature maps are first convolved with a learned kernel and then the results are passed into a nonlinear activation function. We will get different feature maps by applying different kernels. The typical activation functions are sigmoid, tanh and Relu (Nair, Hinton 2010).

Pooling Layer

The sampling process is equivalent to fuzzy filtering. The pooling layer has the effect of the secondary feature extraction, it can reduce the dimensions of the feature maps and increase the robustness of feature extraction. It is usually placed between two Convolutional layers. The size of feature maps in pooling layer is determined according to the moving step of kernels. The typical pooling operations are average pooling (Wang et al. 2012) and max pooling (Boureau et al. 2010). We can extract the high-level characteristics of inputs by stacking several Convolutional layer and pooling layer.

Fully-connected layer

In general, the classifier of Convolutional neural network is one or more fully-connected layers. They take all neurons in the previous layer and connect them to every single neuron of current layer. There is no spatial information preserved in fully-connected layers. The last fully-connected layer is followed by an output layer. For classification tasks, softmax regression is commonly used because of it generating a well-performed probability distribution of the outputs. Another commonly used method is SVM, which can be combined with CNNs to solve different classification tasks (Tang 2013).

Because a fully connected layer occupies most of the parameters, it is prone to overfitting. One method to reduce overfitting is dropout (Srivastava et al. 2014). At each training stage, individual nodes are either 'dropped out' of the net with probability $1 - p$ or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed. Only the reduced network is trained on the data in that stage. The removed nodes are then reinserted into the network with their original weights. Also, one more method to prevent overfitting of a network is 'early stopping': to simply stop the training before overfitting has had a chance to occur. But it comes with the disadvantage that the learning process is halted.

Related Work

k-NN based image classification relying was implemented (Amato 2010), which was particularly suited for classifying the images that are described by local features (as opposed to the global features used in object detection tasks), with the proposal of the approach being relied on the possibility of performing similarity search between image local features. With the use of local features generated over interest points, single label k-NN classification approach was revised to consider similarity between local features of the images in the training set rather than similarity between images, thus opening up new opportunities to investigate more efficient and effective strategies.

In the last few years in the field of image classification, the problem of recognizing landmarks has received growing attention by the research community. Methods for placing photos uploaded to Flickr on the world map were presented (Serdyukov et al. 2009). In the proposed approach the images were represented by vectors of features of the tags, and visual keywords derived from a vector quantization of the SIFT (Scale Invariant Feature Transform) descriptors. A combination of context-based and content-based tools were used to generate representative sets of images for location driven features and landmarks (Kennedy, Naaman 2008). In that work, SIFT features were used to establish links between different images which contain views of a single location. However, this information is combined with the textual metadata, while in this project, only content-based classification was considered.

Support Vector Machine (SVM) for histogram based image classification was performed (Chapelle 1999). However, in the case of using SVM for image classification, the existing literature captures only the determination of the parameters for a given value of the regularization, kernel parameters, and the choice of the kernel. In that way, the SVM moves the problem of overfitting from optimizing the parameters stage to the model selection stage, but kernel models can be quite sensitive to overfitting the model selection criterion (Cawley, Talbot 2010). Also, the hinge loss used in the SVM models result in sparsity. However, often the optimal choice of kernel and regularization parameters means one ends up with all the data being support vectors.

Supervised image classification of land-cover categories of geostatistical data was performed by employing AdaBoost (Nishii, Eguchi 2005). This was based on contextual classification on the neighboring pixel, by calculating the posterior probabilities on all the pixels. But for the current problem of image classification in this project, the image data is extracted from raw satellite imagery, and is bound to have a great amount of background noise. Hence, it is disadvantageous to use the Adaboost approach, as it is sensitive to noisy data and outliers, which would result in underfitting and low accuracy of the model.

Classification of remotely sensed imagery using stochastic gradient boosting as a refinement of classification tree analysis (Lawrence et al. 2004) was performed. This work uses stochastic gradient boosting as an improvement over the plain decision tree (classification tree analysis) approach. Although gradient boosting generally performs slightly better than the Random Forest model, there are many hyperparameters to tune, such as the loss function, learning rate, number of estimators, maximum depth of the tree, etc., and it is quite time consuming to tune the gradient boosting algorithm to output its peak performance. Whereas Random Forest is practically tuning-free: the first parameter to choose is the number of estimators. And the more the number of estimators, the better the model, because the multitude of trees serve to reduce the variance. Due to this advantage, Random Forests was picked for this project.

CNNs are very often used in image recognition, video analysis and natural language processing tasks. In 2012 an error rate of 0.23 percent on the MNIST database was reported (Ciresan et al. 2012). Another paper on using CNN for image classification reported that the learning process was ‘surprisingly fast’; in the same paper, the best published results as of 2011 were achieved in the MNIST database and the NORB database (Ciresan et al. 2011). When applied to facial recognition, CNNs achieved a large decrease in error rate (Lawrence et al. 1997). Also, another research work reported a 97.6 percent recognition rate on 5,600 still images of more than 10 subjects (Matusugu et al. 2003). However, the best implementations of CNNs still struggled with recognizing objects that are small or thin, such as a small ant on the stem of a flower. They also faced trouble with images that have been distorted with filters, an increasingly common phenomenon with modern digital cameras. By contrast, those kinds of images rarely trouble humans. Humans, however, tend to have trouble with other issues; for example, they are not good at classifying objects into fine-grained categories such as the particular breed of dog or species of bird, whereas CNNs handle this.

Project Description

This section is dedicated to explaining the pipeline of the project which includes: data description, data preprocessing, and implementation of the models.

Data description

There are two datasets—training and test—which are available in the JSON format. There are 1604 images in the training dataset, out of which 753 images are labeled as an iceberg, and 851 images are labeled as a ship. Each image in the training data has:

1. id: Identifier for the image.

2. band_1, band_2: The flattened image data. Each band has 75×75 pixel values in the list, so the list has 5625 elements. These values are not the conventional non-negative integral RGB values in the image files, since they have physical meanings—these are float numbers with unit being decibels (dB). Band 1 and Band 2 are the signals characterized by radar backscatter produced from different polarizations at a particular incidence angle. The polarizations correspond to HH (transmit/receive horizontally) and HV (transmit horizontally and receive vertically).
3. inc_angle: The incidence angle of which the image was taken. This field has missing data marked as 'na', and those images with 'na' incidence angles are all in the training data to prevent leakage.

Data preprocessing

4 different and independent (as opposed to sequential) data preprocessing steps were considered for this project. For the ease of understanding of their usage in the experiments section, they are termed as data configurations: D1, D2, D3 and D4. These configurations describe the structure of the data to be input as training data for the models.

D1

In this data configuration, the feature inc_angle, which had 133 missing values out of the total 1604 entries in the training data, is removed. All these rows were in class ship. Because there should not be any missing values to be fed as input to the classification models, this feature was removed. The shape of the data for D1 is (1604, 11250).

D2

In this data configuration, all the rows with missing value ('na') for the feature inc_angle are dropped from the training data. There are a total of 133 such rows and all of them belong to the class ship, as seen in D1. After their removal, there are 753 entries classified as an iceberg, and 718 entries classified as a ship. The shape of the data set for D2 is (1471, 11251).

D3

In this configuration, both the bands band_1 and band_2 are merged together, and the rows with missing values for the feature inc_angle are removed, as done in D2. Basically, the two bands (11250 elements each) have been merged into one band (5625 elements). Because the two bands are in log scale (decibels), the exponential value for each element was taken, then both these exponential values were added. Finally, the log of the sum obtained in the previous step was taken, to form the new band:

$$\text{band}_{\text{new}} = \log_{10}\left(10^{\text{band}_1} + 10^{\text{band}_2}\right)$$

The shape of the data set for D3 is (1471, 5626).

D4

In this data configuration, the decibel values present in band_1 and band_2 are normalized to fit on a scale of 0 to 1. The metric used for this normalization is:

$$X_{\text{normalized}} = \frac{(X_{\text{I}} - \min(X))}{(\max(X) - \min(X))}$$

The scaled bands are then reshaped to 75×75 matrices. Band_1 and Band_2 images were merged into single matrices of size 75x75x2 to create a dual channel image. This was repeated for all training (1604) and testing (8424) inputs.

Model architecture

The architecture for all the models (and their variants) is discussed here.

k-Nearest Neighbors model

The working details of how the k-nearest neighbors algorithm can be used for the classification task was covered in the background section. Two weight functions have been experimented with: *uniform weights* (all the points in the neighborhood are weighed equally) and *distance* (weight points by the inverse of their distance, i.e., closer neighbors of a query point will have a greater influence than neighbors which are further away). The algorithm used for computing the nearest neighbors is *k*-d tree (space-partitioning data structure for organizing points in a k-dimensional space). The power parameter (p) of 2 was used for the Minkowski metric. The Minkowski distance of order p between two points $X = (x_1, x_2, \dots, x_n)$ and $Y = (y_1, y_2, \dots, y_n) \in \mathbb{R}^n$ is defined as:

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

p=2 makes the distance metric as Euclidean distance. Finally, the model is tested with a variable number of nearest neighbors to consider (k), in the range {1, 3, 5, 7, 9, 11, 13, 15, 17}.

Random Forests model

The working details of the random forests model was discussed in the background section. The criterion function used is 'gini', which determines how the quality of a split (node of a decision tree) is measured. The number of maximum features to consider when looking for the best split is taken as the square root of the total number of features:

$$\text{max\_features\_split} = \sqrt{n(\text{features})}$$

The minimum number of samples required to split an internal node is considered as 32. No specific limit was set for the maximum depth of the tree. So, the nodes are expanded until all leaves are pure or until all leaves contain less than 32 samples. The minimum number of samples required to be at a leaf node is taken as 1. Bootstrap samples are used to

build the tree. Out-of-bag samples to estimate the generalization accuracy was not used.

Convolutional Neural Networks model-1

The first model was built based on a basic CNN. The model was built with a batch size of 32, which defines the number of samples propagated through the network per batch. Also, the CNN was implemented with a 3x3 convolutional filter, and 2x2 pooling filter. It has a convolutional depth of 32 kernels or neurons in the first convolutional layer, and 64 kernels in the second convolutional layer. The model has two dropout layers: first with the probability of 0.2 and the second with 0.65. Lastly, the model has a fully-connected layer consisting of 512 fully-connected neurons. All convolutional layers used the 'same' padding method and a 'relu' activation function. The model consisted of a total of 10,683,650 parameters.

Convolutional Neural Networks model-2

The second varies from the first model (model-1) in terms of the model hyperparameters. Model-2 had a smaller batch size of 16. The CNN also was implemented with a 3x3 convolutional filter, and 2x2 pooling filter. This model consisted of three convolutional layers, the first two with 32 kernels and the last convolutional layer with 64 kernels. There is a single drop out layer with a 0.5 drop out the probability and a fully connected layer of 512 neurons. Much like CNN model-1, model-2 also uses a 'relu' activation function. The model consisted of a total of 229,473 parameters.

Experiments

This section deals with the experiments conducted in the project, and these experiments are organized according to the model. All the models were trained using 10-fold cross validation.

Out of the 4 data configurations seen in the data preprocessing step of the previous section, the last configuration D4 is fed as input exclusively only for the CNN models. The k-NN and Random Forests models are given inputs only from the data configurations D1, D2 and D3. Hence, whenever an experiment is conducted using either the k-NN model or the Random Forests model, all the inputs from the 3 non-CNN data configurations: D1, D2 and D3, are tried separately, and their results are compared with one another. In that manner, the most efficient data preprocessing step will also be uncovered, apart from the best performing model.

k-Nearest Neighbors model

The k-NN model was experimented twice, by varying the weight functions in each experiment. In turn, in each experiment, the value for the number of nearest neighbors (k) were varied in the previously specified range of {1, 3, 5, 7, 9, 11, 13, 15, 17}. In the first experiment, the model was given weight function as *uniform*, which means, upon finding out all the k nearest neighbors for a given value of k, all such neighbors are weighted equally, without preference toward any of those neighbors. The results of this experiment are tabulated and presented in table 1.

k	D1	D2	D3
1	0.733	0.749	0.701
3	0.747	0.758	0.707
5	0.746	0.751	0.697
7	0.745	0.745	0.703
9	0.739	0.750	0.684
11	0.735	0.742	0.689
13	0.728	0.731	0.686
15	0.724	0.727	0.684
17	0.724	0.727	0.683

Table 1: Performance (accuracy scores) of k-NN model using uniform weights function.

In the second experiment, the model was given the weight function as *distance*, which means that weights are formed in an inversely proportional relation with the distance of the neighbors from a test data point. So, the nearer neighbors are given preferred to the neighbors further away. The results for this experiment are tabulated and presented in table 2, where the accuracy scores are indicated for each data configuration, as the value of k increases from 1 to 17 in an alternative-skip sequence.

k	D1	D2	D3
1	0.736	0.750	0.701
3	0.749	0.759	0.707
5	0.746	0.752	0.697
7	0.745	0.746	0.703
8	0.742	0.751	0.684
11	0.737	0.743	0.689
13	0.729	0.732	0.686
15	0.725	0.728	0.684
17	0.724	0.727	0.683

Table 1: Performance (accuracy scores) of k-NN model using distance weights function.

Random Forests model

The intention of exploring Random Forests model was to observe the expected improvement in the performance (accuracy score) as the number of estimators used (decision trees used internally) are increased. With the same motivation, this experiment was designed to train the Random Forest model discussed in the model architecture sub-section of

the previous section, on all the 3 available data configurations D1, D2 and D3, by increasing the number of estimators in each round of the experiment. Initially started at n=4, the number of estimators (n) were gradually increased in powers of 2, up to 11 (i.e., $2^{11} = 2056$ trees). The results of this experiment are tabulated and presented in table 3.

n	D1	D2	D3
4	0.668	0.672	0.641
8	0.695	0.698	0.658
16	0.718	0.722	0.672
32	0.725	0.735	0.685
64	0.738	0.748	0.703
128	0.749	0.762	0.718
256	0.757	0.779	0.727
512	0.782	0.791	0.747
1024	0.798	0.803	0.774
2056	0.808	0.817	0.79.1

Table 3: Performance (accuracy scores) analysis of Random Forests model by varying the number of estimator (n).

Convolutional Neural Network model-1

Using a 10 fold cross-validation method, an approximate validation accuracy rate of 87 percent was obtained by the CNN model-1 at 25 epochs. Model-1 had an approximate validation loss of about 29 percent. The resultant plots for this model are shown in figure 1. Considering the relative simplicity of the model, it can be said that the model performed relatively well as it outperformed most of the other methods tried so far.

Convolutional Neural Network model-2

The most successful model among all was CNN model-2, which achieved an approximate validation accuracy rate of 89 percent was obtained by the CNN model-2 at 37 epochs. Model-2 had an approximate validation loss of about 27 percent. The resultant plots for this model are shown in figure 2. Considering the relative simplicity of the model, it can be said that the model performed quite well as it outperformed all the other methods tried so far.

Analysis

For the k-NN model, the distance weight function seemed to have slightly outperformed the uniform weight function. This can be understood, in terms of our image features, that when the test image is being compared with k-nearest neighboring samples, preferring the nearer neighbors to the further away neighbors seems to be advantageous. The highest accuracy obtained by the uniform weight function was 75.8% for input D2, whereas the highest accuracy obtained by the distance weight function was 75.9%, which is only a slight improvement over the uniform weight function.

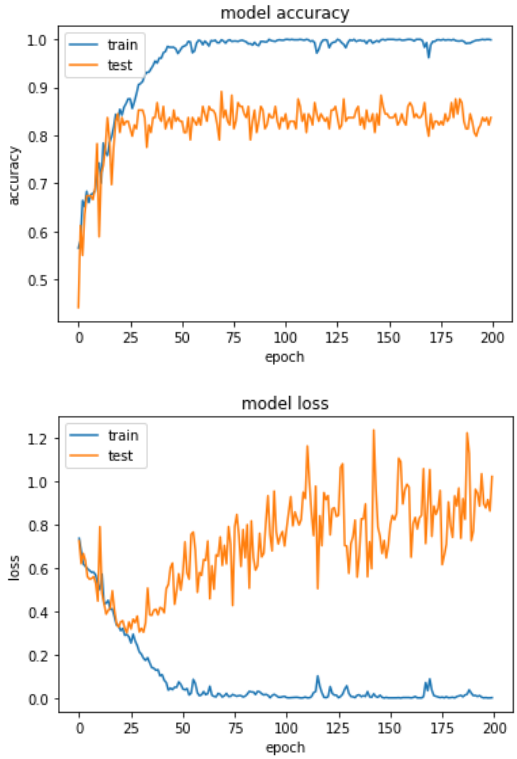

Figure 1: Performance analysis of CNN model-1. The top plot shows model accuracy and the bottom plot shows model loss.

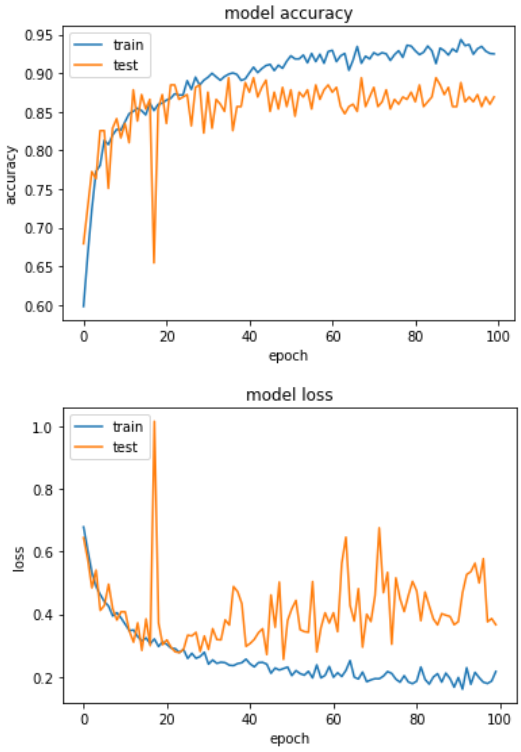

Figure 2: Performance analysis of CNN model-2. The top plot shows model accuracy and the bottom plot shows model loss.

For both the uniform and distance weight functions of the k-NN model, the highest accuracies were obtained for the number of nearest neighbors (k) = 3. As the experiment involved testing with multiple values of k, it was expected that at higher values of k the model would overfit the data, hence, the validation accuracies declined as the value of k grew.

The Random Forest model performed expectedly well with the highest accuracy of 81.7% for the data configuration D2 and with the number of estimators as 2056. It can be clearly observed from the table that as the value of the number of estimators in the random forests model grew, the total variance produced by all the decision tree in the forest, is minimized, and hence the accuracy scores improved as the estimators grew.

Finally, the convolutional neural network model have the highest accuracies among all the models tried in this project. The model-2 of CNN which obtained an approximate validation accuracy of 89% serves as the best model for the task of classifying images, in the context of distinguishing icebergs from ships. The only differences between the architectures of CNN model-1 and CNN model-2 are the number of convolutional layers and the number of dropout layers. Also, the batch size was smaller for the model-2. With the effect of adding a convolutional layer of 32 neurons and removing a dropout layer, and offsetting to a probability of 0.5, the model-2 performed approximately 2% more efficiently on the validation dataset.

Conclusion

Three models have been explored in this project that were tried on the StatOil/C-Core iceberg dataset. Although the k-Nearest Neighbors algorithm is basic and simple, it was experimented with two different weight functions: uniform and distance. It was observed that the distance weight function performed slightly better than the uniform weight function model of k-NN. The best accuracy for the k-NN model was obtained for the value k=3, for the input data where all the rows with missing values for the incidence angle feature were dropped. Random Forests model performed expectedly well with a high accuracy of 81.7% on the same input as of the k-NN model's peak. It was observed that as the number of estimators grew in the forest, the variance produced by the trees was reduced and resulted in a better accuracy of classification. Finally, the experiments done in this project serve well to prove that Convolutional Neural Networks can obtain a great accuracy for image recognition/classification. The highest accuracy of 89% was obtained by the CNN model-2 and the next best was 87%, obtained by the CNN model-1.

References

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. and Berg, A.C., 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3), pp.211-252.

Amato, Giuseppe & Falchi, Fabrizio. (2010). KNN based image classification relying on local feature similarity. *Proceedings - 3rd International Conference on SImilarity Search and Applications*, SISAP 2010. 101-108. 10.1145/1862344.1862360.

D. Coomans; D.L. Massart (1982). Alternative k-nearest neighbour rules in supervised pattern recognition: Part 1. k-Nearest neighbour classification by using alternative voting rules. *Analytica Chimica Acta*. 136: 15–27

Altman, N. S. (1992). An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*. 46 (3): 175–185.

Everitt, B. S., Landau, S., Leese, M. and Stahl, D. (2011) Miscellaneous Clustering Methods, in Cluster Analysis, 5th Edition, *John Wiley & Sons, Ltd, Chichester*, UK.

Ho, Tin Kam (1995). Random Decision Forests. *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, Montreal, QC, 14–16 August 1995. pp. 278–282.

Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2008). *The Elements of Statistical Learning* (2nd ed.). Springer. ISBN 0-387-95284-5.

Ho, Tin Kam (2002). A Data Complexity Analysis of Comparative Advantages of Decision Forest Constructors. *Pattern Analysis and Applications*: 102–112.

LeCun, Yann. LeNet-5, convolutional neural networks. Retrieved 16 November 2013.

V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines, in *ICML*, 2010, pp. 807–814.

T. Wang, D. Wu, A. Coates, and A. Ng, End-to-end text recognition with Convolutionalal neural networks, in *International Conference on Pattern Recognition* (ICPR), 2012, pp. 3304–3308.

Y. Boureau, J. Ponce, and Y. Le Cun, A theoretical analysis of feature pooling in visual recognition, in *ICML*, 2010, pp. 111–18

Y. Tang, Deep learning using linear support vector machines, arXiv prprint arXiv:1306.0239, 2013.

Srivastava, Nitish; C. Geoffrey Hinton; Alex Krizhevsky; Ilya Sutskever; Ruslan Salakhutdinov (2014). Dropout: A Simple Way to Prevent Neural Networks from overfitting. *Journal of Machine Learning Research.* 15 (1): 1929–1958.

P. Serdyukov, V. Murdock, and R. van Zwol. Placing Flickr photos on a map. In SIGIR '09: *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 484{491, New York, NY, USA, 2009. ACM.

Chapelle, O., Haffner, P. and Vapnik, V.N., 1999. Support vector machines for histogram-based image classification. *IEEE transactions on Neural Networks*, 10(5), pp.1055-1064.

G. C. Cawley and N. L. C. Talbot, Over-fitting in model selection and subsequent selection bias in performance evaluation, *Journal of Machine Learning Research*, 2010. Research, vol. 11, pp. 2079-2107, July 2010.

Nishii, R. and Eguchi, S., 2005. Supervised image classification by contextual AdaBoost based on posteriors in neighborhoods. *IEEE Transactions on Geoscience and Remote Sensing*, 43(11), pp.2547-2554.

Ciresan, Dan; Meier, Ueli; Schmidhuber, Jürgen (June 2012). Multi-column deep neural networks for image classification. 2012 *IEEE Conference on Computer Vision and Pattern Recognition.* New York, NY: Institute of Electrical and Electronics Engineers (IEEE): 3642–3649.

Ciresan, Dan; Ueli Meier; Jonathan Masci; Luca M. Gambardella; Jurgen Schmidhuber (2011). Flexible, High Performance Convolutional Neural Networks for Image Classification. *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*-Volume Volume Two. 2: 1237–1242.

Lawrence, Steve; C. Lee Giles; Ah Chung Tsoi; Andrew D. Back (1997). Face Recognition: A Convolutional Neural Network Approach. *Neural Networks, IEEE Transactions* on. 8 (1): 98–113.

Matusugu, Masakazu; Katsuhiko Mori; Yusuke Mitari; Yuji Kaneda (2003). Subject independent facial expression recognition with robust face detection using a convolutional neural network. *Neural Networks*. 16 (5): 555–559.